# AbstractApplicationContext

## prepareBeanFactory

beanFactory.setBeanClassLoader(getClassLoader())**;**beanFactory.setBeanExpressionResolver(new StandardBeanExpressionResolver(beanFactory.getBeanClassLoader()))**;**beanFactory.addPropertyEditorRegistrar(new ResourceEditorRegistrar(this**,** getEnvironment()))**;**

### 添加BeanPostProcessor

beanFactory.addBeanPostProcessor(new ApplicationContextAwareProcessor(this))**;** // 第1个BeanPostProcessor

beanFactory.addBeanPostProcessor(new ApplicationListenerDetector(this))**;** // 第2个BeanPostProcessor

### 织入

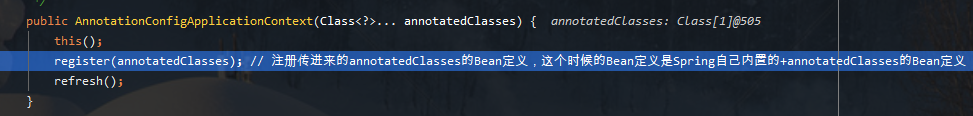
如果有添加

// Detect a LoadTimeWeaver and prepare for weaving, if found.  
if (beanFactory.containsBean(*LOAD\_TIME\_WEAVER\_BEAN\_NAME*)) {  
 beanFactory.addBeanPostProcessor(new LoadTimeWeaverAwareProcessor(beanFactory))**;** // Set a temporary ClassLoader for type matching.  
 beanFactory.setTempClassLoader(new ContextTypeMatchClassLoader(beanFactory.getBeanClassLoader()))**;**}

### 注册3个环境bean

// Register default environment beans.  
if (!beanFactory.containsLocalBean(*ENVIRONMENT\_BEAN\_NAME*)) {  
 beanFactory.registerSingleton(*ENVIRONMENT\_BEAN\_NAME***,** getEnvironment())**;** // 注册环境Bean StandardEnvironment  
}  
if (!beanFactory.containsLocalBean(*SYSTEM\_PROPERTIES\_BEAN\_NAME*)) {  
 beanFactory.registerSingleton(*SYSTEM\_PROPERTIES\_BEAN\_NAME***,** getEnvironment().getSystemProperties())**;** // 注册系统参数Bean  
}  
if (!beanFactory.containsLocalBean(*SYSTEM\_ENVIRONMENT\_BEAN\_NAME*)) {  
 beanFactory.registerSingleton(*SYSTEM\_ENVIRONMENT\_BEAN\_NAME***,** getEnvironment().getSystemEnvironment())**;** // 注册环境变量Bean  
}

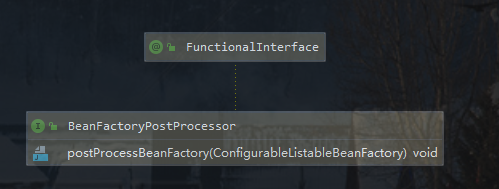
# AnnotationConfigApplicationContext



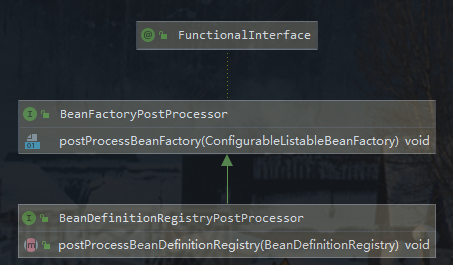
## register

注册Bean定义

# BeanFactoryPostProcessor



## BeanDefinitionRegistryPostProcessor



## 具体的BeanFactoryPostProcessor

### BeanDefinitionRegistryPostProcessor

这个是BeanFactoryPostProcessor特殊的一种

void postProcessBeanDefinitionRegistry(BeanDefinitionRegistry registry) throws BeansException**;**

#### ConfigurationClassPostProcessor

**第一步:**

先调用postProcessBeanDefinitionRegistry，这个方法会调用processConfigBeanDefinitions

这里会得到当前Bean定义中，所有标注@Configuration的类，然后循环解析

### ConfigurationClassPostProcessor

org.springframework.context.annotation.internalConfigurationAnnotationProcessor

### AutowiredAnnotationBeanPostProcessor

org.springframework.context.annotation.internalAutowiredAnnotationProcessor

### CommonAnnotationBeanPostProcessor

org.springframework.context.annotation.internalCommonAnnotationProcessor

### EventListenerMethodProcessor

org.springframework.context.event.internalEventListenerProcessor

### DefaultEventListenerFactory

org.springframework.context.event.internalEventListenerFactory

## registerPostProcessor

上面几个BeanFactoryPostProcessor都是通过registerPostProcessor注册

// role: BeanDefinition.ROLE\_INFRASTRUCTURE  
private static BeanDefinitionHolder registerPostProcessor(  
 BeanDefinitionRegistry registry**,** RootBeanDefinition definition**,** String beanName) {  
  
 definition.setRole(BeanDefinition.*ROLE\_INFRASTRUCTURE*)**;** // InfrastructureAdvisorAutoProxyCreator只会为基础设施类型的Advisor自动创建代理对象  
 registry.registerBeanDefinition(beanName**,** definition)**;**//org.springframework.context.annotation.AnnotationConfigApplicationContext  
 return new BeanDefinitionHolder(definition**,** beanName)**;**}

# BeanPostProcessor

## ApplicationContextAwareProcessor

## ApplicationListenerDetector

## LoadTimeWeaverAwareProcessor

有weaver才会添加

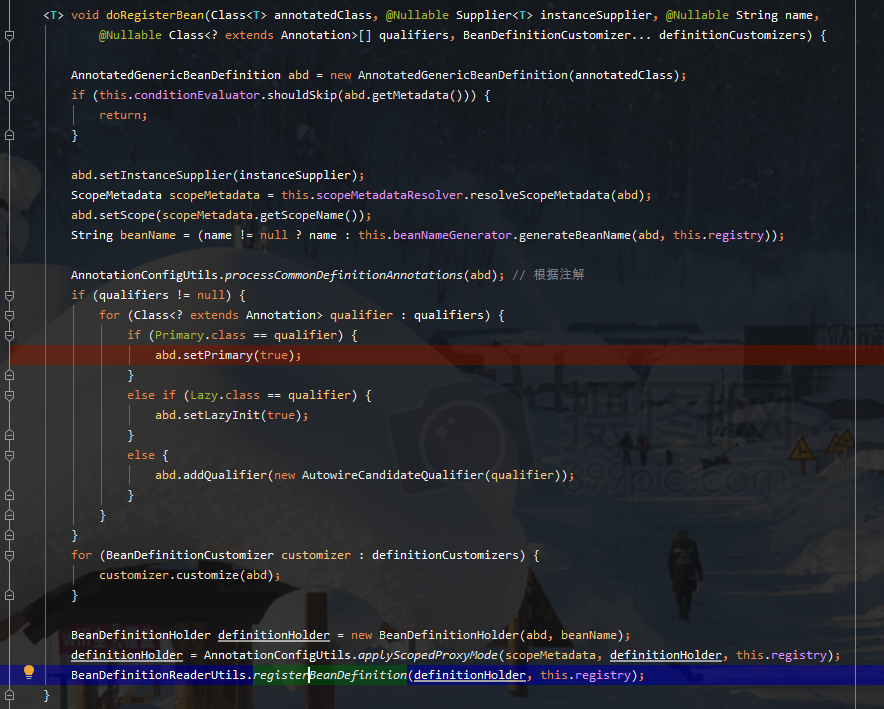
# AnnotatedBeanDefinitionReader

这个类初始化的时候，会把BeanFactoryPostProcessor的Bean定义都加载

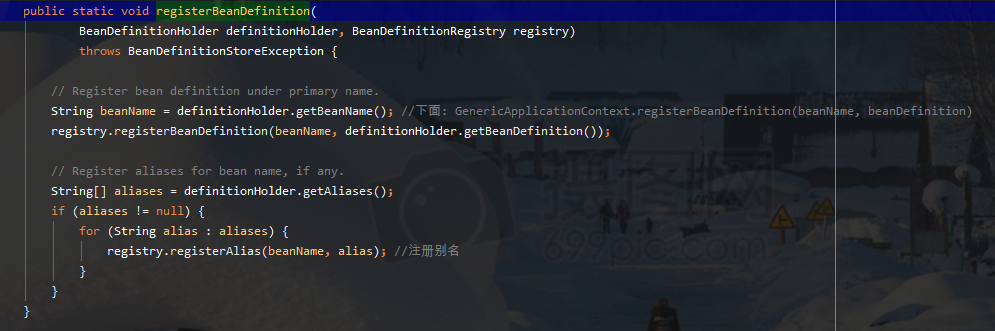
## register

public void register(Class<?>... annotatedClasses) { // 比如一些Configuration  
 for (Class<?> annotatedClass : annotatedClasses) {  
 registerBean(annotatedClass)**;** }  
}

### doRegisterBean



### registerBeanDefinition



# ClassPathBeanDefinitionScanner

## 构造函数

### includeFilters

构造函数会添加到includeFilters

Component

javax.annotation.ManagedBean（默认没有）

javax.inject.Named (默认没有）

# PostProcessorRegistrationDelegate

## invokeBeanFactoryPostProcessors

先实例化BeanDefinitionRegistryPostProcessor

# 解析

## ConfigurationClassParser

解析@Configuration